Electronic Version
Stylesheet Version v1.1.1

Description

METHOD FOR PROGRAM DEBUGGING

BACKGROUND OF INVENTION

[0001] 1. Field of the Invention

[0002] The present invention relates to a method for program debugging, and more particularly, to a method for program debugging which can debug program codes.

[0003] 2. Description of the Prior Art

[0004] In the programming field, the development tools are low level program languages and high level program languages. The text of program codes written in low level languages (e.g. assembly language) is more similar to machine language, but the programming and debugging are not easy. Due to better executing efficiency, low level languages are usually used in basic input/output systems (BIOS) or driver programs.

[0005] Take a BIOS for example. After a computer turns on, the main function of the BIOS is to a execute power on self test (POST) to detect whether the settings of peripheral

devices are correct, peripheral devices being controlled by the operating system. With the complexity of computers, types of peripheral devices are gradually increasing, such as keyboards, disc units, disc controllers, hard disks, video disc players and so on. Therefore the operation of BIOS becomes more complex. Suppose that the operation of a device on the motherboard is improper or that the programming of BIOS program code contains an error. Since the operating system is not loaded yet, errors are debugged by source codes instead of debug software. This requires more time and a lot of work.

[0006] To speed up the debug process, there are two general methods. One method is to insert a debug interface card into the motherboard and to add several diagnosis codes to the BIOS program code. When the BIOS program code is executed at different sections, a diagnosis code would be sent from port 80h to the debug interface card for locking the section. Therefore sections at which errors occur can be found by trapping diagnosis codes.

[0007] Please refer to Fig. 1. The other method is called in circuit emulator (ICE). Take a host 7 as the program start, and connect the host 7 and an emulator 8 by the connection port of the host 7 (RS232 or printer parallel port). Also,

take a motherboard 9 as the program target, and connect the motherboard 9 and the emulator 8. The host includes assemblers, script files, and so on. The emulator 8 is utilized to simulate a CPU and the emulator 8 is inserted into the CPU slot. Therefore the software of the host 7 can be utilized to control the emulator 8 for remote debugging.

[0008] Although the remote debugging provides a good environment for software and hardware, the debug process is still executed by technicians, who execute the script file to BIOS and observe if BIOS is executed as expected. A technician needs to set many breakpoints in places where bugs might occur, and when running into a breakpoint, stop executing the program and determine if the result of the execution is expected. If not, this means that there are bugs in the program codes and the technician must narrow the range to execute the same steps to search for program errors.

[0009] However, in the debug of BIOS, besides errors of the program codes, the most important thing is how error handlers of BIOS deal with errors of peripheral devices. For instance, when BIOS detects hardware errors, the motherboard typically provides an alert such as a beep. Due to a large number of peripheral devices, the debug method

mentioned above cannot detect if error handlers are missed, and cannot debug all of the error handlers immediately.

Summary of Invention

[0010] It is therefore a primary objective of the claimed invention to provide a method which can immediately debug error handlers of an implementation under test (IUT).

[0011] The method for program debugging in the claimed invention comprises: setting a plurality of breakpoints corresponding to a plurality of events in an implementation under test, executing the implementation under test for outputting a diagnosis code of a breakpoint, resetting a parameter of the event corresponding to the diagnosis code, and executing the event according to the reset parameter for making the event undergo an error handler.

[0012] These and other objectives of the present invention will no doubt become obvious to those of ordinary skill in the art after reading the following detailed description of the preferred embodiment that is illustrated in the various figures and drawings.

Brief Description of Drawings

[0013] Fig. 1 is an ICE debug system.

[0014] Fig. 2 shows steps of the method for program debugging of the present invention.

[0015] Fig. 3 shows addresses of breakpoints in an implementation under test.

[0016] Fig. 4 shows steps of the script file of the present invention.

[0017] Fig. 5 shows steps of the implementation under test of the present invention.

Detailed Description

[0018] The above-mentioned and other techniques and advantages of the present invention are clearly disclosed in the following embodiment of the present invention.

[0019] Before describing the present invention, it should be noted that the present invention is still executed by an ICE. Development tools for programming script files and emulators connected to the host can be selected from ITP products such as those of American Arium or Cisco Systems, which are known to one of ordinary skill in the art.

[0020] Please refer to Fig. 2. Step 21 recites setting a plurality of breakpoints corresponding to a plurality of events in an IUT. In the embodiment of the present invention, take a BIOS for example. Of course, the IUT can be a driver program or other similar program.

[0021] Please refer to Fig. 3. Since the BIOS program code executes BIOS for peripheral devices, the BIOS program code can be divided into many test modules. Since there are many modules in BIOS, consider a CMOS test module, a memory test module, and a hardware test module for example. Events are defined as tests executed to each peripheral device. In each event, there are two processing states: one is a general processing state when the device is working well and the other is an error processing state when the device is out of order.

[0022] A breakpoint is set corresponding to an event, as shown in Table 1. Diagnosis codes are defined by designers. Note that addresses of each breakpoint can be set ahead of each event. Also, addresses of each breakpoint can be set after each event.

Table 1

Breakpoint	Diagnosis code	Test
1st breakpoint	00h	CMOS test
2nd breakpoint	01h	Memory test
:	:	:
N-th breakpoint	N-1h	Hardware test

[0023] The interrupt nature of breakpoints means a software in-

terrupt, using a trap to stop the execution of the emulator. In addition, in the embodiment of the present invention, the output port of the above-mentioned diagnosis codes is port 90h.

[0024] Next, as step 22 recites, the IUT is executed by means of the script file for outputting different diagnosis codes at different sections, such as outputting 01h.

[0025] Step 23 recites that in each event, the branch of instructions is executed after assigning different parameters; therefore, step 23 includes two sub-steps: (1) deleting parameters of corresponding events in the IUT, and (2) inputting new parameters for turning the path of general state into the path of error state. For instance, if the above-mentioned diagnosis code 01h is detected, the original parameter of the IUT is placed with 0FAh. This causes the execution of memory test module to turn into the path of error handler "no memory".

[0026] Finally, as step 24 recites, after resetting the parameter in step 23, when the program code of the corresponding event is executed, this can simulate the hardware under an error condition, and the program code can execute the error handler. For instance, an error message could be displayed, a signal to make the motherboard emit an au-

dible signal (beep) could be generated, or a system reset or stopping signal could be generated. Take the above-mentioned diagnosis code 01h for example. In step 24, it sends a signal of Beep 2-1-2 to inform users of a memory error state. As mentioned above, if the breakpoints are set ahead of the events, the parameters are reset after the breakpoints. If the breakpoints are set after the events, the script file should spontaneously jump ahead of the branch command of the event and reset the parameter of the event.

[0027] Therefore, by the above-mentioned method, when executing steps 23 and 24 to each event, error states of IUT can be detected one by one. On the premise that we do not have to destroy the hardware, we can know if the ability of IUT to handle the error state when the hardware is out of order is sufficient.

[0028] Furthermore, as shown in Fig. 4, users also can debug a certain event of the IUT, such as hardware error. Step 41 recites selecting a certain event. Next, step 42 recites executing the IUT. Since the breakpoints are set in the IUT already, if the IUT is executed at the breakpoints instead of the end of the program, then the diagnosis codes of the breakpoints are inspected, as in step 43 and 44. When

the execution of all of the commands is correct, the script file is written with the result of the execution of the IUT into for history tracking, as in step 45.

[0029] Step 46 recites that the diagnosis code, when the program is interrupted, is compared with the diagnosis code which a user assigns. If the diagnosis code answers to the diagnosis code which a user assigns, step 47 is executed for resetting the related parameters. If the diagnosis code does not answer to the diagnosis code that the user assigned, the IUT is executed until the diagnosis code answers to the diagnosis code that the user assigned.

[0030] Please refer to Fig. 5. Step 51 recites that due to the script file, the IUT outputs different diagnosis codes while executing different sections and stops outputting when the diagnosis code answers to the user assigned diagnosis code.

[0031] Step 52 recites that when the diagnosis code answers to the user assigned diagnosis code, the event under test is being executed and the parameters of the IUT are deleted and reset by the script file. If the reset parameters are the same as the original parameters, the execution of the IUT is the path of general state. If the reset parameters are different from the original parameters, the execution of

the IUT is the path of error state. There are two error handlers: one is the generic event error handler and the other is the critical event error handler, both being respectively executed according to different parameters. The generic event error handler displays error messages and writes errors into a file. The critical event error handler is an audible tone (Beep 3-3-2, Beep 2-2-1, and so on.), a system reset, or a stop execution command, and so on. Finally, after undergoing the error handler, the IUT is executed until the end of the program.

[0032] The above-mentioned steps are executed by means of the above-mentioned program tools according to the breakpoints for executing the script file, and the IUT is debugged automatically.

[0033] Therefore, by the above-mentioned steps, whether selecting a certain event under test or testing all of the events in an IUT, both can be executed speedily. Whats more, execute error handlers can be executed without destroying the hardware, and after the IUT is amended and recompiled, debugging can be continued without rewriting the script file. This can achieve the purpose of the present invention.

[0034] Those skilled in the art will readily observe that numerous

modifications and alterations of the device may be made while retaining the teachings of the invention. Accordingly, the above disclosure should be construed as limited only by the metes and bounds of the appended claims.